## 一：背景介绍

ijkplayer的Ff\_ffplay的是参考ffmpeg里的ffplay来实现的，构建API提供给ijkplayer使用，ijkplayer就是ijkmediaplayer的核心实现API。

播放入口stream\_open，里面实现了两个核心的线程read\_thread和video\_refresh。Ff\_ffplayer额外增加实现了SDL\_Aout音频输出和SDL\_Vout视频渲染两个逻辑处理。

video\_refresh是处理解码后的Video的数据如何进行渲染（Audio的数据渲染是在audio\_open）。

read\_thread是读取原始数据并解码，先利用ffmpeg的Demuxer和Parser来分离出Audio/Video流，再通过stream\_component\_open来处理不同的流，对于Audio, 创建了Audio\_thread解码线程，对于Video，创建了Video\_thread解码线程。

FFPlayer里维护着一个VideoState，管理着不同的Audio/Video流的信息，audio\_st和video\_st两个AVStream，audioq和videoq两个PacketQueue， 以及auddec和viddec两个Decoder。

开始播放，进行数据缓冲，缓冲数据信息分别存储在audioq和videoq里，缓冲过程中，Audio\_thread和Video\_thread并行的获取数据进行解码，当发现从auddec和viddec里面获取不到数据时，也会触发数据缓冲。

因此，流媒体的缓冲流程机制其核心就在于VideoState的使用实现。

## 二：APP层获取缓冲信息的API

##### 1. setOnInfoListener

注册IMediaPlayer.OnInfoListener，在onInfo里：

MEDIA\_INFO\_BUFFERING\_START //开始缓冲

MEDIA\_INFO\_BUFFERING\_END //缓冲结束

该事件由ffp\_toggle\_buffering接口来触发，参数1表示开始缓冲，参数0表示结束缓冲。

**MEDIA\_INFO\_BUFFERING\_START**开始缓冲事件的触发点：

1> FFP\_REQ\_START的时候，也就是mediaplayer的start时触发

2> read\_thread线程里面的for循环过程中，如果判断seek\_req为true，也就是需要seek的时候，触发

3> decoder\_decode\_frame解码过程中，如果发现没有数据也就是从auddec和viddec里面获取不到数据，会触发

**MEDIA\_INFO\_BUFFERING\_END**缓冲结束事件的触发的地方：

1> read\_thread线程里面的for循环过程中，发现audioq和videoq的size总和超过FFPlayer里的dcc（FFDemuxCacheControl）的max\_buffer\_size时触发

2> read\_thread线程里面的for循环过程中,当所有的数据都渲染完了，也就是sampq和pictq里的rindex\_shown等于size的时候会触发

3> read\_thread线程里面的for循环过程中,读取数据失败，发现是eof了，触发

4> ffp\_check\_buffering\_l确认buffering状态时，如果发现流closed或者abort的时候触发

**我们的视频质量监控采集主要依赖的是这两个缓冲开始，缓冲结束的事件。根据当前测试下来，未发现异常，后续继续跟踪。**

##### 2. setOnBufferingUpdateListener

注册IMediaPlayer.OnBufferingUpdateListener，在onBufferingUpdate里：

percent //缓冲进度条，百分比值

这个百分比值是指当前缓冲到的时间点与资源总时间的比值。

底层实现在ffp\_check\_buffering\_l，依赖的就是VideoState的使用。

取video\_cache和audio\_cache两个Cache时间值的最小值，加上当前的时间值ffp\_get\_current\_position\_l来得到当前的缓冲时间点。

percent就是这个缓冲时间点/总的时间长度（VideoState里的ic里的duration）。

ijkplayer提供了四个接口，可以在APP监控Audio/Video的Cache情况：

**getVideoCachedDuration**

获取Video Stream的Cache缓冲进度（时间单位ms）

**getAudioCachedDuration**

获取Audio Stream的Cache缓冲进度 （时间单位ms）

**getVideoCachedBytes**

获取Video Stream的Cache缓冲大小 （字节单位 Byte）

**getAudioCachedBytes**

获取Audio Stream的Cache缓冲大小 （字节单位 Byte）

这四个接口的数据信息都是来源于FFPlayer里面的stat(FFStatistic)里保存着的video\_cache和audio\_cache（FFTrackCacheStatistic）两个Cache信息。

## 三：VideoState的缓冲数据管理

##### 1. 几个关键变量的意义：

**AVInputFormat \*iformat;**

stream\_open通过avformat\_open\_input接口获取，最早需要获取的资源的format信息，根据这个format信息，我们可以选择合适的ffmpeg的demuxer来解析。demuxer会选择合适的protocol流协议来下载数据流。

**AVFormatContext \*ic;**

同样是在avformat\_open\_input接口里得到。根据匹配的demuxer，然后调用read\_header来建立各个stream信息，存放到streams里，nb\_streams表示有几个stream，同时，还解析出来start\_time（开始时间）， duration（总的时间），bit\_rate（比特率）等。

**PacketQueue audioq;**

**FrameQueue sampq;**

**int audio\_stream;**

**AVStream \*audio\_st;**

**Decoder auddec;**

stream\_open时调用frame\_queue\_init接口初始化了audioq和sampq。audioq保存读取到的Audio的AVPacket信息, sampq保存的读到的Audio的frame信息，frame最大个数是SAMPLE\_QUEUE\_SIZE（9），sampq里面同时也引用保存着audioq。

stream\_component\_open打开Audio流（流在ic里管理）时，根据stream\_index（保存到audio\_stream）得到streams[stream\_index]，保存到audio\_st。调用decoder\_init时构建了auddec，同时，auddec里的queue保存指向着audioq。

sampq主要是在Audio的渲染时用的，decoder\_decode\_frame从auddec里的audioq获取待解码数据，经过解码后，得到Audio的frame, 调用frame\_queue\_push存储到sampq里。在sdl\_audio\_callback，调用audio\_decode\_frame从sampq调用frame\_queue\_peek\_readable获取到frame然后用于SDL audio的渲染。

**PacketQueue videoq;**

**FrameQueue pictq;**

**int video\_stream;**

**AVStream \*video\_st;**

**Decoder viddec;**

stream\_open时调用frame\_queue\_init接口初始化了videoq和pictq。videoq保存读取到的Video的AVPacket信息, pictq保存的读到的Video的frame信息，frame最大个数是VIDEO\_PICTURE\_QUEUE\_SIZE\_DEFAULT（3），pictq里面同时也引用保存着videoq。

stream\_component\_open打开Video流（流在ic里管理）时，根据stream\_index（保存到video\_stream）得到streams[stream\_index]，保存到video\_st。调用decoder\_init时构建了viddec。同时，viddec里的queue保存指向着videoq。

pictq主要是在Video的渲染时用的，decoder\_decode\_frame从viddec里的videoq获取待解码数据，经过解码后，得到Video的frame, 在queue\_picture调用frame\_queue\_push存储到pictq里。在video\_refresh，从pictq调用frame\_queue\_peek\_readable获取到frame然后用于SDL Video的渲染。

##### 2. Audio/Video如何Cache

read\_thread里面的for循环读取数据，调用av\_read\_frame，根据ic(AVFormatContext),最终调用ffmpeg的demuxer的read\_packet接口得到AVPacket数据。

根据AVPacket里的stream\_index是audio\_stream还是video\_stream，分别调用packet\_queue\_put存入audioq或者videoq里。

在FFPlayer的dcc(FFDemuxCacheControl)里面定义了max\_buffer\_size是MAX\_QUEUE\_SIZE（15 \* 1024 \* 1024 ）（缓冲最多不超过15M），也就是audioq.size + videoq.size 不能超过dcc. max\_buffer\_size,超过则触发MEDIA\_INFO\_BUFFERING\_END缓冲结束事件，停止av\_read\_frame。

FFPlayer里面的stat(FFStatistic)里，保存着video\_cache和audio\_cache（FFTrackCacheStatistic）两个Cache信息。Cache信息里面包含三个数据：duration（cache时间），bytes（cache总大小），packets（cache的AVPacket的总个数）。这三个数据都可以分别从audioq和videoq里获取到。

read\_thread里面的for循环每次av\_read\_frame读取到AVPacket数据后，同时会调用ffp\_statistic\_l，也就是从audioq和videoq里获取信息设置到video\_cache和audio\_cache。同时，如果tick时间间隔上次上报时间超过BUFFERING\_CHECK\_PER\_MILLISECONDS（500），会调用ffp\_check\_buffering\_l来更新上报MEDIA\_BUFFERING\_UPDATE事件。